A homomorphic range searching scheme for sensitive data in internet of things

Baohua Huang*, Sheng Liang, Dongdong Xu and Zhuohao Wan

School of Computer and Electronic Information,
Guangxi University,
Nanning, Guangxi 530004, China
Email: bhhuang66@gxu.edu.cn
Email: 3104516237@qq.com
Email: dongdxu@foxmail.com
Email: 355060761@qq.com
*Corresponding author

Abstract: With the popularisation and development of the internet of things, big data and cloud computing, the search of data in cloud-based internet of things becomes a hot research topic. However, the sensitive data, such as the medical data collected by wearable devices, is inevitable to be stored in the cloud server. Homomorphic encryption has the ability to calculate the ciphertext without decryption. We separate the calculating and the decryption into different security domains to preserve the privacy of sensitive data, so the original plaintext would not be exposed in the cloud. Hence, we can compare two ciphertexts to get the difference of them in a privacy preserving way. In order to accelerate the search process of range query, we build an encrypted self-balancing binary index tree. Based on oblivious RAM, the searching scheme can hide the access patterns of the node of tree. The actual nodes and logic relation of tree are stored on different servers. A sample implementation of the proposed scheme is given, and the experimental results and analysis are presented to illustrate the scheme's effectiveness and security.

Keywords: internet of things; range search; homomorphic encryption; oblivious random-access memory.

Reference to this paper should be made as follows: Huang, B., Liang, S., Xu, D. and Wan, Z. (2020) 'A homomorphic range searching scheme for sensitive data in internet of things', *Int. J. Embedded Systems*, Vol. 13, No. 1, pp.101–112.

Biographical notes: Baohua Huang is an Associate Professor and a Graduate Supervisor in the School of Computer and Electronic Information, Guangxi University, China. He received his PhD in Computer Science from Huazhong University of Science and Technology, China, in 2006. He is a senior member of CCF, a member of ACM and IEEE. His research interests include internet of things, vehicular ad hoc network, network security, wireless networks security, database security, homomorphic encryption, encrypted data searching, etc.

Sheng Liang is a postgraduate student of the School of Computer and Electronic Information, Guangxi University, China. His research interests focus on homomorphic encryption and encrypted data searching.

Dongdong Xu is a postgraduate student of the School of Computer and Electronic Information, Guangxi University, China. His research interests focus on database security and encrypted data searching.

Zhuohao Wan is a postgraduate student of the School of Computer and Electronic Information, Guangxi University, China. His research interests focus on database security and encrypted data searching.

This paper is a revised and expanded version of a paper entitled 'A homomorphic searching scheme for sensitive data in NoSQL database' presented at The 11th IEEE International Conference on Cyber, Physical and Social Computing (CPSCom 2018), Halifax, Canada, 30 July to 3 August 2018.

Copyright © 2020 Inderscience Enterprises Ltd.

1 Introduction

The internet of things (IoT) has been attracting much attention from the academic community and industry. According to Gartner, the number of things connected in IoT will reach 50 billion in 2020, as presented by Vermesan and Bacquet (2017). The IoT will play an important role in the future (Alam et al., 2017), because the things could be everywhere to improve our quality of life. The medical IoT (Haghi et al., 2017; Ahmed and Abdallah, 2017) could monitor health indicators in various areas targeted for detection of whether it is healthy or unhealthy.

With the development of cloud computing, many users and enterprises are attracted by the characteristic features, such as pay-as-you-go and scalability. Because of the devices' limit, it's unavoidable to need to store the data in the cloud, which also make the data accessible to other devices and services (Shafagh, 2015). As the cloud computing can provide supporting for the IoT (Roopaei et al., 2017), many users and enterprises would deploy the cloud of things. The medical IoT can improve patients' quality of life and quality of care (Gatouillat et al., 2018), but some data are very sensitive, such as those medical data from wearable devices.

Due to the data were stored in remote servers, the data users lose control of sensitive data. So, data user will encrypt the sensitive data before uploading it to the cloud server. Designing and proving secure and practical encrypted data searching schemes for the IoT domain is still an important open research problem (Shafagh et al., 2015; Kong et al., 2018).

Encryption can preserve the confidentiality of data when outsourcing data to untrusted cloud servers. Ciphertext-policy attribute-based encryption is a suitable technology for enabling fine-grained access control policies on sensitive data in cloud computing (Bai et al., 2017; Shynu and Singh, 2018), while the users could only be able to decrypt the ciphertext that satisfies the access policy. But encryption also introduces the difficulty for query and process on encrypted data, especially for range query. Many range query technologies proposed in the past, such as bucketzation and order-preserving encryption, cannot satisfy both security and efficiency. The bucketzation wastes bandwidth, and it's not suitable for the IoT devices with limited compute resource. And the order-preserving encryption will reveal much sensitive information about order and frequency to cloud server, so this is vulnerable to inference attacks (Islam et al., 2014; Naveed et al., 2015). The Arx (Poddar et al., 2016) uses garbled circuits to protect the order information of ciphertext, but it is necessary to update the garbled circuits that have been used in the past for security.

To overcome the drawbacks mentioned above, we propose a homomorphic range searching scheme for cloud-based IoT sensitive data. Our scheme can protect the privacy of outsourced data stored in semi-trust third party (as honest but curious), and does not reveal ciphertext and relation of it to any server at the same time.

Our contribution: the main contributions of this article can be summarised as follows:

1. We propose a private compare method based on homomorphic encryption by separating calculation and decryption into two different security domains, so the original plaintext does not appear in the system.
2. We propose a secure and effective encrypted index scheme for range search. The order information of ciphertext is not revealed to the *Index Storager*, because ciphertext and relation of it were stored in different security domains, and we use oblivious RAM strategy to hide the access pattern.
3. We implement the above scheme. The experimental results show that our scheme has proved to be secure and effective.

Organisation: the rest of this paper is organised as follows. In Section 2, we introduce some preliminaries that are used in our scheme, and describe the problem modelling. In Section 3, we propose a structure of our homomorphic range index scheme, and describe a simple encrypted range index scheme and an improved scheme, including private compare, inserting and range search procedure. In Section 4, we present the concrete construction of homomorphic range searching and experimental analysis of it. The overview of the related work is presented in Section 5. Finally, we conclude our paper in Section 6.

2 Preliminaries and problem modelling

In this section, we first introduce the notations and some preliminaries involved in our paper, and then define the problem and security.

2.1 Notations

In the rest of paper, some notations are continuously quoted, so the concepts and terminologies were described as following list.

Table 1 The notations used in this paper

Symbol	*Description*
HE	Homomorphic encryption
M	The set of plaintext data
C	The set of ciphertext data
sk	The secret key of cryptosystem
$op, \oplus$	The operation on data
Q	The query
v_c, v^c, or op_c	The operation or value on encrypted statue
c^*, v_c^*	The ciphertext which was perturbed
d	The difference between two value
$data_c'$	The ciphertext of data encrypted with another encryption scheme
r, t, k	The random number

2.2 Homomorphic encryption

The homomorphic encryption, also called as privacy homomorphic, was introduced by Rivest, Adleman and Dertouzous, as described in Gentry (2009). It allows computation on encrypted data without decryption, and decrypts the resulting ciphertext to get plain result finally.

Formally, we define encryption algorithm as *enc* and decryption algorithm as *dec*,

$$enc: M \rightarrow C \tag{1}$$

$$dec: C \rightarrow M \tag{2}$$

And the mapping relationship between operations on plaintext and operations on ciphertext is as follows,

$$\forall op: I \rightarrow 0, enc: I \rightarrow I_c \tag{3}$$

$$\exists op_c: I_c \rightarrow O_c, dec: O_c \rightarrow 0 \tag{4}$$

where I is the set of input of operation *op*, O is the set of output of operation *op*. $I \subset M$, $O \subset M$, $I_c \subset C$, $O_c \subset C$. The op_c is the operation on the ciphertext.

If *enc* and *dec* satisfy the above *formula (1), (2), (3)* and *(4), we can* call the *enc* with *dec* homomorphic encryption.

In short, the operation on ciphertext like this:

$$dec\left(enc\left(m_1\right) \oplus_c enc\left(m_2\right)\right) = m_1 \oplus m_2 \tag{5}$$

There are various types of HE algorithm, including partial HE (PHE), Somewhat HE (SwHE) and Full HE (FHE). If the ⊕ in formula (5) can only be addition or multiplication, it is called PHE because of the limitation of the type of operation supported. If the ⊕ can be any type operation but the operation steps are limited, it is called SwHE. If the type and steps on ⊕ are not limited, it is called FHE.

In order to search the ciphertext, we can use the SwHE or even PHE because it is not necessary to use FHE if the operation steps on searching are limited.

2.3 Oblivious RAM

Oblivious RAM (ORAM) is an efficient strategy for hiding sequences of accesses included reading and writing. It was first introduced by Goldreich and Ostrovsky for software protection, as described in Sánchez-Artigas (2018).

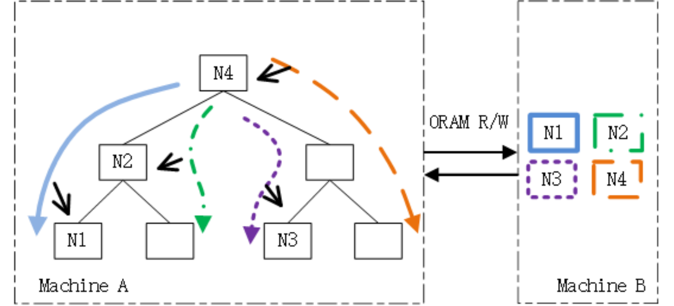
A secure ORAM scheme allows for client protected data stored on untrusted remote server, to hide not only the sensitive data but also the sequence of accesses included reads and writes. The server couldn't realise which data element was read or written, when client convert one operation of read or write into an operation sequence of readings and writings. After obtaining the data, the client renews the cipher and uploads it to the cloud server. So, the server can't learn that whether reading or writing.

An access to d_j can transport to a sequence of read R and write W, like this:

$$Access\left(d_j\right) \rightarrow \begin{Bmatrix} R\left(d_i\right), W\left(d_i\right), \cdots \\ R\left(d_j\right), W\left(d_j\right), \cdots R\left(d_k\right), W\left(d_k\right) \end{Bmatrix} \tag{6}$$

A tree-based ORAM scheme requires $2n - 1$ storage blocks to store n elements, which are respectively stored on a random node in the path from the root to the leaf node. The client will read all nodes in the path from root to leaf, meanwhile, the writing follows the reading of node. A basic tree-based ORAM scheme is showing in Figure 1.

Figure 1 Structure of ORAM scheme (see online version for colours)



A node will store in a random node at the path from root to the leaf. *Machine B* requests some nodes for one node, such as $\log_2 N$ in Figure 1, so *Machine A* can't learn which node is in need for *Machine B*.

2.4 The problem definition

Range search is a retrieval in database that returns all records which match specified range restrictions on some attributes. The range search contains many operators, such as 'x and $x < a$', '$b < x$ and $x \leq a$' and so on, but these operators can be converted into $b - \varepsilon < x < a + \varepsilon$ where ε is a small enough positive number.

The set of records retrieved from database has two features:

1. the specified attribute's value of every record falls in the range
2. the values of these records' specified attributes are linear in numerical value.

In other words, we can accelerate the range search process with order information.

For the range search of ciphertext, we can use the predicate encryption to learn that whether the record satisfies the feature (1) by searching linearly every record. obviously, it is not efficient for huge volume of IoT's sensitive data. The order-preserving encryption reveals the order information and corresponding ciphertext, so range search of ciphertext encrypted with order-preserving encryption can be efficient like searching on plaintext. Another method is to store order information on the index structure.

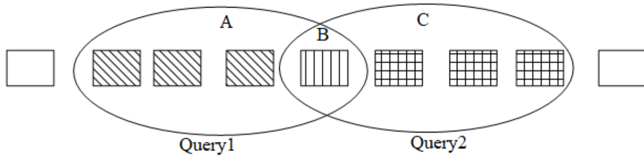
From the functions of encrypted range search, we need to retrieve all records which meet the range restrictions to requester. Meanwhile, we should protect the privacy of sensitive data from adversaries.

2.5 The security definition

The secure range search aims to protect the privacy of sensitive data from adversaries. We assume that adversary is semi-trusted third party, i.e., the adversaries are curious about sensitive data but behave according to protocol, it is also called honest-but-curious. Meanwhile, we also assume that there is no collusion from two different security domains.

The order information can accelerate the search process, but it is vulnerable to inference attacks. Even if the order information was not directly exposed to cloud, it is also vulnerable to the access analysis on the static index structure. The order information can be inferred from the history of range search, as described by Wong et al. (2017). Figure 2 show that adversaries can get the order information from history of query.

Figure 2 Intersection of query results



If adversary can obtain the intersection B of two different query results A and C, the partial order relation will be exposed, such as $A \leq B \leq C$. If adversary can learn enough intersections of query results, the order information of all ciphertext will be revealed to adversary. In addition, the access pattern will also reveal the relation of data that it is important for index structure which saved the order information. However, we need to know the order information to accelerate the range query process.

From the security of encrypted range search, we need to encrypt the data and avoid revealing the ciphertext and order information of it, including resisting the access analysis.

3 Homomorphic searching scheme based on random-read method

In this section, we describe the system structure, including private compare, inserting and range search.

3.1 Structure of the scheme

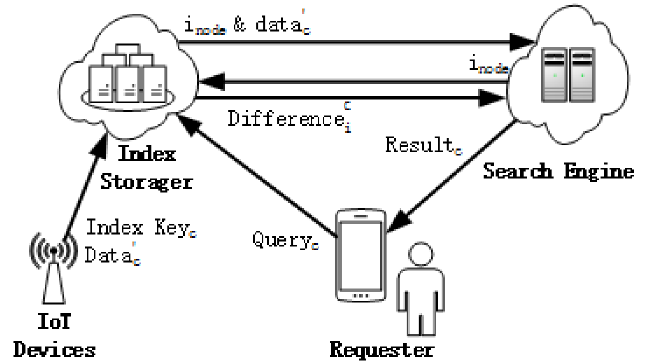
In order to separate calculation and decryption of searching procedure, we can design the structure of the scheme as shown in Figure 3.

The structure of homomorphic searching is shown in Figure 3, which consists of the following entities.

- *IoT devices* encrypt sensitive index attribute and other data with homomorphic encryption and another encryption (such as AES) correspondingly.
- *Index Storager* provides storage service for IoT devices' sensitive attribute and responds on difference queries from *Search Engine*. For the $Data_c'$, *Index Storager* just is a forwarder without other processing.
- *Search Engine* stores the relation of index attributes' ciphertext encrypted with homomorphic encryption, such as binary index tree, and other data encrypted by AES, and responds on range queries from *Requester*.
- *Requester* encrypts the range and send to *Storager*.

Sensitive index attributes' values were encrypted with homomorphic encryption and stored in cloud database. The other data was encrypted with another encryption scheme or another secret key. So, *Storager* cannot learn that which index value corresponds to which data. The *Storager* and *Search Engine* are two services running in two different security domains. The original ciphertext and the decryption key cannot appear in the same security domain, the system cannot learn the original plaintext.

Figure 3 Structure of homomorphic searching scheme



3.2 Private compare procedure

Since we use the encrypted binary tree to preserve the order information, we should have an ability to compare the ciphertext. With the help of homomorphic encryption, we can calculate the difference between the query value and index value without decryption. So, we can get the compare result, such as $<_c$, $=_c$ and $>_c$.

The private compare operation is shown as below:

$$d_c^* = (a_c -_c b_c) \times_c r_c \tag{7}$$

$$dec: d_c^* \rightarrow d \times r \tag{8}$$

The x_c is the ciphertext of a random positive number. The multiplication with a positive number has no effect on the sign of number, therefore the perturbation operation does not change the compare result. Suppose we need the all of data greater than the q_c, if d equals or greater than zero, then the data index by key_c and its right subtree is fall in our query range and vice versa.

In the comparing procedure, neither key_c nor q_c has been decrypted as we just need to know which is greater. The adversary cannot recovery the plaintext from the perturbed compare result d_c^*, although he holds the decryption key sk. The private compare process is shown in Algorithm 1.

Algorithm 1 Private Compare Process

1 *Search Engine* sends position (i, j) to *Index Storager*
2 Storager send $(k_i^c -_c k_j^c) \times_c r_c$ to *Search Engine*
3 *Search Engine* decrypt the ciphertext
4 If the plaintext of difference is greater than 0
5 *Search Engine* learn that $k_i > k_j$
6 End if
7 If the plaintext is lesser than 0
8 *Search Engine* learn that $k_i < k_j$
9 End if
10 If the plaintext is equal to 0
11 *Search Engine* learn that $k_i = k_j$
12 End if

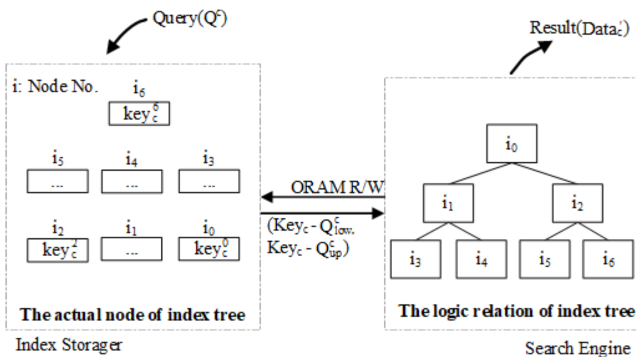
To avoid exposing statistical properties, we should ensure this *difference = 0* does not appear which contain some sensitive information that query is equal to some ciphertexts or the new node inserted to index tree is equal to some nodes. Therefore, we can subtract a small enough positive number ε for the low boundary of search range or add ε for the up boundary of it.

Through such processing, the *Search Engine* holding decryption key cannot access the original ciphertext, such as key_c and q_c, but can search on encrypted binary index tree.

3.3 Separation of relation from ciphertext

In order to accelerate the search process, we build a binary index tree for the sensitive data, shown in Figure 4. The *Index Storager* stores the actual nodes' value of sensitive attribute and send the $data_c'$ to *Search Engine*. The $data_c'$ and the index key_c values cannot be decrypted by the same decryption key. The *Search Engine* server stores the relation of index value in a self-balancing binary tree without exact plaintext of index value. When the *Requester* looks for data in the query range, the *Search Engine* will get and decrypt the difference between key value of node and value of query range from *Index Storager* server to decide traverse path on index tree.

Figure 4 Structure of indexing tree



The operation of range search on this encrypted self-balancing binary tree is like the unencrypted self-balancing binary tree, except that the compare process between node's value and range's value for the search engine does not directly hold those ciphertext. So, the compare operation should be instead of the private compare showed in Algorithm 1.

But the access patterns will reveal the relation of nodes, which leak the structure of the index tree to *Index Storager*. If we do not hide the access sequence to *Index Storager*, for example, the first one node of access sequence must be the root node and the second node of access sequence must be the child node of root node. When the adversaries monitor enough access sequence, they can learn that all of child nodes to parent node, then even all relations of nodes in the index tree. So, we need to transform the access sequence into a random-access sequence.

Based on ORAM described as Figure 1, the pseudocode of range searching procedure is shown in Algorithm 2, and the statements are executed by the *Requester*, *Index Storager* or *Search Engine*.

Algorithm 2 Range Search procedure based on ORAM

1 *Requester* send (Q_{low}^c, Q_{up}^c) to *Index Storager*
2 *Search Engine* set the $node_{curr}$ from root node
3 While the $node_{curr}$ is not null
4 Get $Q_{low}^c - key_{curr}^c$, $Q_{up}^c - key_{curr}^c$ and $Data_{curr}^{c'}$ from Storager by ORAM
5 Decrypt the difference
6 If the $node_{curr}$ in the range
7 Add $Data_{curr}^{c'}$ into ResultSet
8 End if
9 If $node_{curr}$ in the left of range
10 Search right subtreerecursively
11 End if
12 If $node_{curr}$ in the right of range
13 Search left subtree recursively
14 End if
15 If $node_{curr}$ in the range
16 If parent is in range and $node_{curr}$ is left child
17 Add right subtree into ResultSet
18 Search left subtree recursively
19 End if
20 If parent is in range and $node_{curr}$ is right child
21 Add left subtree into ResultSet
22 Search right subtreerecursively
23 End if
24 If parent node not in the range
25 Search left and right subtreerecursively
26 End if
27 End if
28 End while
29 *Search Engine* send ResultSetto *Requester*

Although the oblivious RAM described above can hide the access pattern, it is not efficient for the huge volume of data.

3.4 A random-read for hiding access pattern

Although the relationship of ciphertexts and ciphertext are stored in different servers separately, if the access pattern is exposed to the adversary, some sensitive information will be exposed by inference attack. Tree-based ORAM has a high cost in terms of storage and communication, so we can use a random read method to hide the trust access pattern.

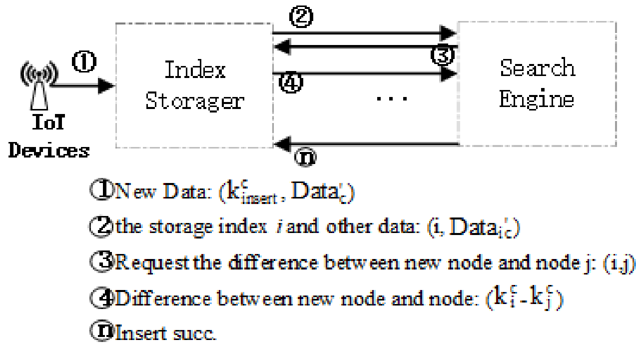
Search Engine can randomly select a node and compare it with the query range. If the node does not fall in the range, *Search Engine* will filter out the nodes and subtrees that are not in the range, because he knows the order of data. *Search Engine* select a node from those rest nodes and recursive the above operation until finding all the nodes in the range.

3.4.1 Inserting procedure

When inserting a new data, we should maintain the index tree for storing the order information to accelerate the range search process. So, the inserting procedure needs to know the position of the new node on the tree.

There's something different about inserting a new node into a normal self-balancing binary tree for the comparison between a new node and a node of the tree come from Index Storage, and we should hide the relation of nodes from *Index Storager*. As mentioned above, we can introduce a small enough positive number for avoiding the difference being equal to 0. The detail of inserting procedure is shown in Figure 5.

Figure 5 Inserting procedure for encrypted index tree



The IoT device will encrypt the sensitive attribute with HE and other data will be encrypted with another encryption like AES. The encrypted data will be sent to *Index Storager*, then *Index Storager* will send it to *Search Engine*. Because *Search Engine* needs to know which position is appropriate for the new data for preserve the order information, *Index Storager* should return the difference between the new node and some nodes which are randomly picked to *Search Engine*.

As mentioned above, we can exclude some nodes or subtree from the index tree by randomly comparing a node and search range. For inserting a new node into a self-balancing binary tree, we should know which subtree is the appropriate subtree according to the procedure is described in Algorithm 3.

Algorithm 3 Random Access the Appropriate Subtree

Find an appropriate subtree from a binary tree

1 Copy a tree that its depth is *n* from the original index tree
2 While do not get the appropriate subtree
3 Get a random node from tree
4 Private compare between the random node and the new node
5 If the random node is lesser than the new node
6 If the random node is the maximum
7 The right child is the appropriate subtree
8 End if
9 Clear those nodes in outside of range
10 End if
11 If the random node is greater than the new node
12 If the random node is the minimum
13 The left child is the appropriate subtree
14 End if
15 Clear those nodes in outside of range
16 End if
17 If the random node is equal to the new node
18 Clear those nodes in outside of range
19 End if
20 End while

The pseudocode of inserting procedure for new index key is shown in Algorithm 4, and the statements are executed by the IoT Terminal, *Index Storager* or *Search Engine*.

Algorithm 4 Inserting New Node into the Binary Index Tree

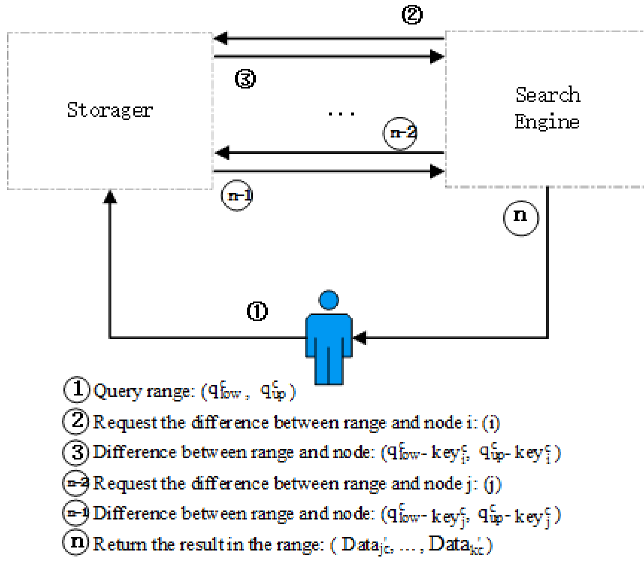
1 IoT Terminal sends a new encrypted data to *Storager*
2 *Search Engine* set root into subtree
3 Random access the appropriate subtree
4 While the appropriate subtree is a not-null subtree
5 Get the appropriate subtree by the random access or conventional access
6 Set the appropriate subtree into root
7 End while
8 Put the new node into the appropriate position
9 Maintain the balancing of tree

The inserting procedure chooses an appropriate node of subtree as described in Algorithm 3. We can randomly choose a node from a tree that its depth is shorter than the original tree, so *Index Storager* does not get the knowledge about which one is the root node. The greater the depth is, the more random the access will be. In other words, the first node accessed to index tree can be any one in the highest *n* layer of index tree. Due to the randomness, *Index Storager* cannot learn that which one is another one's child node.

3.4.2 Range searching procedure

For the range query, we should know the relation of range and the key of an index node, such as left, in or right. We can know whether the query range is intersected with the left or right subtree of the current node with the homomorphic compare. When the parent node and the current node are all in the query range, if the current node is the parent's left/right child, the right/left subtree must in the query range. The detail of range searching procedure is shown in Figure 6.

Figure 6 Searching procedure for range query (see online version for colours)



If we know some compare information about those nodes in the binary index tree, we can learn that which nodes are or not in the search range that is shown in Algorithm 5.

Algorithm 5 Getting nodes and subtrees in range	
1	While do not learn that the compare result for some nodes to search range
2	Get a random node from these nodes
3	If the random node is lesser than the range
4	Clear those nodes and subtrees that are lesser than the random node
5	End if
6	If the random node is greater than the range
7	Clear those nodes and subtrees that are greater than the random node
8	End if
9	If the random node is in the range
10	Mark the random
11	If the other node is in the range
12	Mark those nodes that are lesser than one and greater than another
13	End if
14	End if
15	End while

Like the inserting procedure, we should randomly read the node from index tree for hiding the relation of nodes. The pseudocode of range searching procedure is shown in Algorithm 6, the statements are executed by executed by the *Requester*, *Index Storager* or *Search Engine*.

Algorithm 6 A Range Searching Procedure with random access	
1	*Requester* sends (Q^c_{low}, Q^c_{up}) to *Index Storager*
2	*Search Engine* copy a tree that its depth is *t* from the original index tree
3	While there are some
4	Getting nodes and subtrees in range
5	Add these nodes into ResultSet
6	Add those subtrees into ResultSet, except the left subtree of minimum and the right subtree of maximum
7	The minimal left subtree and the maximal right subtree can be search by conventional search or the random-search recursively
8	End while
9	Send the ResultSet and *QR* to *Requester*

The above procedure can support the inserting and range, and the deletion can be derived from range search.

3.5 Analysis of security and effectiveness

Because of encryption, perturbation and separation of computation and decryption, the original plaintexts do not occur in the system. *Index Storager* can access the ciphertext but cannot decrypt it due to the lack of the decryption key. At the same time, access patterns od index data are also hidden by *Search Engine* with adding fake, random access. The *Search Engine* can learn nothing but the difference perturbed by a random positive number. In addition, because of diffusion and confusion, the $data'_c$ encrypted by another encryption scheme can resist the adversary's attack.

Table 2 The compare on different range search scheme

Scheme	*Insert time*	*Search time*	*Storage*	*Involving client*
Linearly search	1	N	N	No
Based-AES OPE	$\text{Log}_2 N$	$\text{Log}_2 N$	N	Yes
Based-tree ORAM	$(\text{Log}_2 N)^2$	$(\text{Log}_2 N)^2$	$2N - 1$	No
Our scheme	$\text{Log}_2 N$	$\text{Log}_2 N$	N	No

In order to avoid reveal of order information by access pattern, the ORAM can hide the access pattern well. But it needs to cost more storage capacity and burden of bandwidth. In addition, the client needs to insert $\log_2 N - 1$ fake node with the insertion of one trust node for ORAM. Our scheme can achieve logarithmic search efficiency with the order information, when neglecting the influence of random factors. The details of insertion, search, and storage are shown in Table 2. Our scheme does not require

additional storage capacity and does not require the client to involve in the search process. If the communication on different security domains takes more time than decryption, the random-read search can get the result set more quickly.

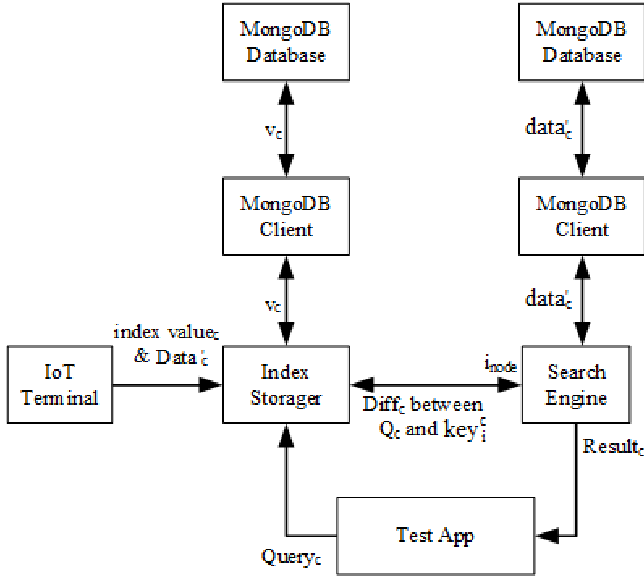
4 Experimental

In this section, we will describe the experimental conditions, explain the concrete implementation for a simple somewhat homomorphic encryption, and give the experimental results and analysis.

4.1 Experimental environment

Based on our previous transparent middleware, we can store and access the encrypted data in MongoDB and it facilitates the access encrypted index data and other encrypted data for the *Search Engine*. Figure 7 gave the architecture of our sample system.

Figure 7 Architecture of the sample system



Index Storager need to store some sensitive attribute and *Search Engine* need to store the order information of sensitive attribute and other encrypted data, these data were encrypted by other encryption scheme, so the two servers in different security domains use the MongoDB to store encrypted data correspondingly.

Because it is needs to know the relation of those index node, the *Search Engine* should have the ability to decrypt the ciphertext of difference. *Search Engine* just has the decrypt key for the ciphertext difference, not for the encrypted $Data'_c$.

The running environment is: Workstation with 8 GB RAM, one Intel Xeon CPU E5-1620 with 8 cores, and Ubuntu 18 operating system.

For comparison, we have implemented the AES-based order preserving encoding, proposed by Popa et al. (2013) and the two-cloud linear search scheme, proposed by Xue et al. (2017). The homomorphic encryption's key is 2,048 bits, and the AES's key is 128 bits.

4.2 Homomorphic encryption algorithm

A simple implementation for somewhat homomorphic encryption was provided by Gentry (2009). Sun et al. (2018) improved the scheme from 1 bit to n bit. In our sample implementation of the homomorphic range searching scheme, we use the symmetric homomorphic encryption. We randomly select an odd integer p *with system parameter* λ. We encrypt the n bit b by adding a random multiple of p and a random multiple of 2^n.

The homomorphic encryption includes at least three functions:

- $GenKey(\lambda) \to key$: input the security parameter λ, output the odd integer p as the secret key.
- $Enc(key, m) \to c$: input the key p and plaintext m, output the ciphertext c:

  $$c = m + 2^n r + kp \tag{9}$$

  where r and k are some integers chosen from certain range.
- $Dec(key, c) \to m$: input the key p and ciphertext c, output the plaintext m:

  $$m = (c \bmod p) \bmod 2^n \tag{10}$$

  if $m + 2^n r$ falls in $(-p/2, p/2)$, the decryption will correctly output the plaintext m.

Actually, (c mod p), which is the noise parameter in this scheme, will be in $(-p/2, p/2)$, since $m + 2^n r$ is in that range.

The decryption would have worked correctly if satisfy the formula (12).

$$b + 2^m r \in \left(-\frac{p}{2}, \frac{p}{2}\right) \tag{11}$$

According to formula (9), multi steps of calculation on ciphertext may lead to decryption error. If we need one step of subtraction, the m and r should satisfy the following condition:

$$-2^{n-1} < m - (-m) < 2^{n-1} \tag{12}$$

$$-p/2 < 2m + 2^{n+1} r < p/2 \tag{13}$$

Combine the formula (12) and (13):

$$\begin{cases} -p/2 < 2^{n-1} + 2^{n+1} r < p/2 \\ -p/2 < -2^{n-1} + 2^{n+1} r < p/2 \end{cases} \tag{14}$$

$$\left(\frac{-p}{2^n} + 1\right) \Big/ 4 < r < \left(\frac{p}{2^n} - 1\right) \Big/ 4 \tag{15}$$

After subtraction operation, the difference is multiplied by a positive number t to perturb the compare result, then the value range of r is:

$$\begin{cases} -p/2 < \left(2^{n-1} + 2^{n+1} r\right) \cdot t < p/2 \\ -p/2 < \left(-2^{n-1} + 2^{n+1} r\right) \cdot t < p/2 \end{cases} \tag{16}$$

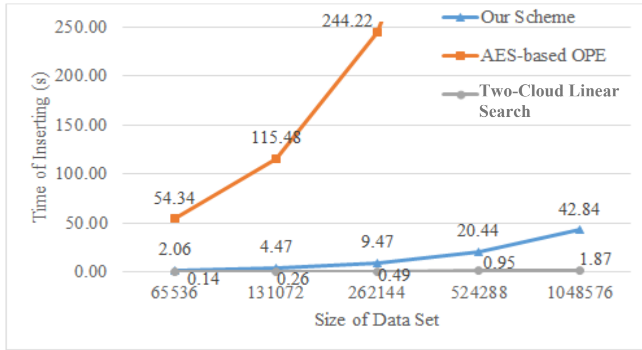
$$\left(\frac{-p}{2^n \cdot t} + 1\right) \Big/ 4 < r < \left(\frac{p}{2^n \cdot t} - 1\right) \Big/ 4 \tag{17}$$

Like above, we can calculate the ciphertext without decryption error by choosing parameters carefully. Our program is based on a free library GMP for arbitrary precision arithmetic.

4.3 Experimental result

First of all, we observe the time cost of item insert under various conditions. To avoid interference from network's complexity, we ignore the time network transmission. And they will be connected with high speed network. We will explain the number of network transfers needed. The result of inserting data is shown in Figure 8.

Figure 8 Time of inserting different size of data set (see online version for colours)

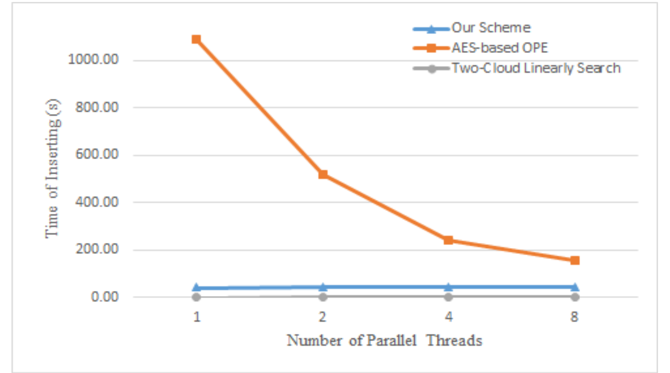


The two-cloud linear search scheme performs better, because the insert operation just is encryption and not to preserve the order information of cipher. The time consumed in two-cloud linear search has a linear relationship with size of data set. The AES encrypts every new node into 128 bits, and the simple somewhat homomorphic encryption like formula (12) needs less time than AES. Figure 8 show that the time consumed in AES-based OPE and our scheme had logarithmic relationship with size of data set, because the depth of index tree is $\log_2 N$ where N is the size of data set.

The parallel processes should be able to accelerate the insert operation. So, we conducted experiments in a multithreading environment that the size of data set is 1,048,576 as described in Figure 9.

From Figure 9, we can see that the multithread accelerates the AES-based OPE scheme. Compared to AES-based OPE, the simple homomorphic encryption used by our scheme and two-cloud linear search takes less time. So, the acceleration of AES-based OPE on different numbers of parallel threads is more obvious.

Figure 9 Time of inserting encrypted datawith different numbers of parallel threads (see online version for colours)

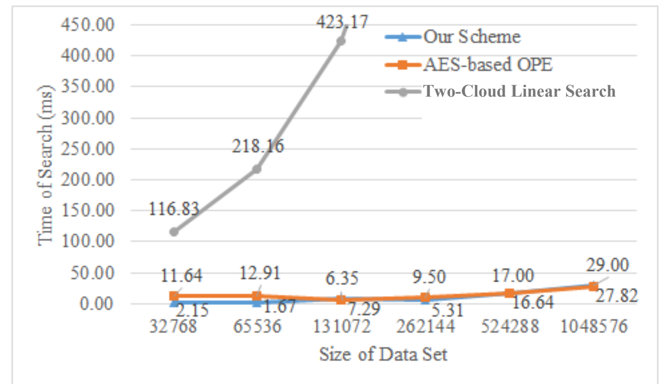


Note: Size of data set is 1,048,576.

Although the time of index-based insert is greater than linear insert, index-based search should perform better than linear search for the order of encrypted data were preserved.

For the property of self-balancing binary tree, insert double data into index tree just increases the depth of tree by 1. So, the search time would not grow rapidly with the increase of data set's size. The experimental search result is shown in Figure 10.

Figure 10 Searching time of different data set's size (see online version for colours)



With the increase of the size of data set, the time of linear search also rapidly increases. We can see that:

1 the time of tree index searching is lower than that of linear searching

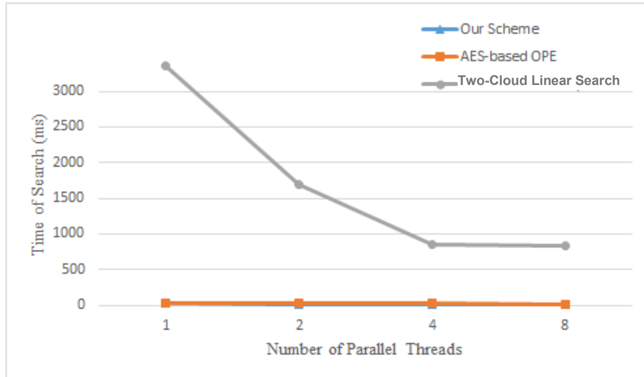
2 the time of linear searching has a linear correlation with the size of data set

3 the time of index-based searching has a logarithmic relationship with the size of data set.

It is clear that the index-based search performs better than the linear search for the index-based scheme preserve the order information.

Like the insert operation, we also conducted experiments under conditions of different threads number. The result is shown in Figure 11.

Figure 11 Time of search encrypted data with different number of parallel threads (see online version for colours)

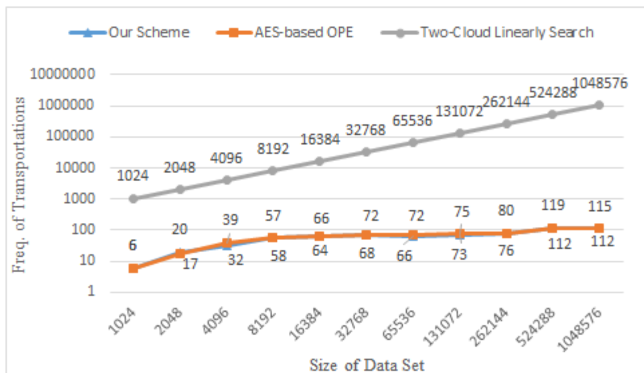


Note: Size of data set is 1,048,576.

Unlike insert operations, index-based search performs better in experiments, because the index-based search just needs to compare $\log_2 N$ data node. The linear search scheme need to compare all data with query, so the accelerated effect look better for parallel.

Even if the time of linear search is equal to that of index-based search as the number of parallel threads increases, the transportation required for linear search is obviously higher than that for index-based search. For security reasons, the calculating and decryption were separated into two hosts in different domain. At the same time, we need to hide the access pattern. So, we observe the frequency of communication in two different security domains. The frequency of communication is shown in Figure 12.

Figure 12 Transport frequency of different size of data set (see online version for colours)



From Figure 12, we can see that our scheme and OPE scheme need less transportation than the two-cloud linear search scheme. For the huge volume of IoT data, frequent communication will significantly aggravate the burden on networks. Compared with OPE scheme, our scheme does not need a fully trusted requester involved in the insert and search processes.

5 Related works

The rapid development of IoT brings with it the unavoidable issue about data management for the huge volume of IoT data. Some researchers are devoted to the study of power efficiency, as described by Zhang et al. (2015) and Paethong et al. (2016), cost-efficient, as described by Xie (2017) of low-level data storage, etc. There are also lot of works on high level data management. In the previous work, Tian et al. (2014) proposed and implemented a transparent middleware for encryption data in MongoDB. Many researchers have also done a lot of work on data storage, sharing and searching for the encrypted IoT data.

Amghar et al. (2018) discuss the main task for IoT data storage and management, and compare the populate NoSQL Database. But the security issues mentioned on this paper are mainly aimed at the external attackers. Jiang et al. (2014) propose a data storage framework for IoT, which can efficiently store the huge volume IoT data. It can handle structured and unstructured data by combining and extending multiple databases and Hadoop. But the main concern of this paper is the functionality of data storage in IoT.

Thirumalai and Shanmugam (2017) propose a communication scheme based on an encryption algorithm for protecting the secure communication among things. Shafagh et al. (2017) introduce construction for sharing of IoT data by fully homomorphic encryption, it can re-encrypt the ciphertext under the Alice's key into the ciphertext under the Bob's key without decryption.

PrivacyProtector, proposed by Luo et al. (2018), is a framework stored patients' private data by secret sharing. The patients' private data was divided into some secrets, which were stored on some different servers.

Yang et al. (2019) proposed a file storage system for IoT-based healthcare big data. It can share the medical file among the healthcare staff from different medical domains by a cross-domain access control policy. For the characteristics of medical event, the staff can access the data by a proper attribute secret key.

Wu et al. (2018) propose a searchable encryption for the cloud-based military IoT using the trapdoor permutation function. Fu et al. (2018) design a framework to handle the time-sensitive and non-time-sensitive IoT data. The non-time-sensitive data is stored in the cloud server, and it can support ciphertext search by secure kNN algorithm.

The Talos, proposed by Shafagh et al. (2015a), can not only store the IoT data but also allows to query processing on ciphertext. Like CryptDB provided by Popa et al. (2011), the Talos allows query processing on ciphertext with a combination of order-preserving encryption, homomorphic encryption and the other schemes.

6 Conclusions

There are few sensitive data searching schemes that can meet the needs of both security and efficiency as far as we know. The searchable encryption protocol is too complex, and the order-preserving encryption is vulnerable to inference attacks. It's possible to calculate on ciphertext and get encrypted result without decryption by homomorphic encryption. The homomorphic range search can preserve the privacy of original ciphertext by separating the calculation and decryption into different security domains. Based on the homomorphic compare, we build an encrypted binary index tree to accelerate the search process of range query. The *Index Storager* stores the actual encrypted value of node with ORAM and the *Search Engine* knows the relation of node. The *Index Storager* cannot get the plaintext from ciphertext for it hasn't the decryption key. Even though holding the decryption key, the *Search Engine* can't get the original ciphertext. In addition, the actual value and relation of nodes are not revealed to any server at the same time.

In the future, we will develop a secure and efficient multiuser homomorphic encryption and improve the parallelism for reading from *Index Storager*.

Acknowledgements

This work was supported by National Natural Science Foundation of China under grant No. 61262072.

References

Ahmed, A.B. and Abdallah, A.B. (2017) 'Architecture and design of real-time system for elderly health monitoring', *Int. J. Embedded Systems*, Vol. 9, No. 5, pp.484–494.

Alam, F., Mehmood, R., Katib, I., Albogami, N. N. and Albeshri, A. (2017) 'Data fusion and IoT for smart ubiquitous environments: a survey', *IEEE Access*, April, Vol. 5, pp.9533–9554.

Amghar, S., Cherdal, S. and Mouline, S. (2018) 'Which NoSQL database for IoT Applications?', *MoWNet: 2018 International Conference on Selected Topics in Mobile and Wireless Networking*, Tangier, Morocco, pp.131–137.

Bai, C., Zhang, Y., Ma, H. and Liu, Z. (2017) 'Expressive ciphertext-policy attribute-based encryption with direct user revocation', *Int. J. Embedded Systems*, Vol. 9, No. 6, pp.495–504.

Fu, J., Liu, Y., Chao, H.C., Bhargava, B. and Zhang, Z. (2018) 'Secure data storage and searching for industrial IoT by integrating fog computing and cloud computing', *IEEE Transactions on Industrial Informatics*, Vol. 14, No. 10, p.4519–4528.

Gatouillat, A., Massot, B., Badr, Y., Sejdić, E. and Gehin, C. (2018) 'Building IoT-enabled wearable medical devices: an application to a wearable, multiparametric, cardiorespiratory sensor', *BIOSTEC 2018: Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies*, Funchal, Madeira, Portugal, pp.109–118.

Gentry, C. (2009) *A Fully Homomorphic Encryption Scheme*, PhD thesis, Stanford University, Palo Alto, USA.

Haghi, M., Thurow, K. and Stoll, R. (2017) 'Wearable devices in medical internet of things: scientific research and commercially available devices', *Healthcare Informatics Research*, Vol. 23, No. 1, pp.4–15.

Islam, M. S., Kuzu, M. and Kantarcioglu, M. (2014) 'Inference attack against encrypted range queries on outsourced databases', *CODASPY 2014: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, San Antonio, Texas, USA, pp.235–246.

Jiang, L., Da Xu, L., Cai, H., Jiang, Z., Bu, F. and Xu, B. (2014) 'An IoT-oriented data storage framework in cloud computing platform', *IEEE Transactions on Industrial Informatics*, Vol. 10, No. 2, pp.1443–1451.

Kong, W., Lei, Y. and Ma, J. (2018) 'Data security and privacy information challenges in cloud computing', *Int. J. Computational Science and Engineering*, Vol. 16, No. 3, pp.215–218.

Luo, E., Bhuiyan, M.Z.A., Wang, G., Rahman, M.A., Wu, J. and Atiquzzaman, M. (2018) 'Privacy protector: privacy-protected patient data collection in IoT-based healthcare systems', *IEEE Communications Magazine*, Vol. 56, No. 2, pp.163–168.

Naveed, M., Kamara, S. and Wright, C. V. (2015) 'Inference attacks on property-preserving encrypted databases', *ACM CCS 2015: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, Colorado, USA, pp.644–655.

Paethong, P., Sato, M. and Namiki, M. (2016) 'Low-power distributed NoSQL database for IoT middleware', *ICT-ISPC 2016: Proceedings of 2016 Fifth ICT International Student Project Conference*, Nakhon pathom, Thailand, pp.158–161.

Poddar, R., Boelter, T. and Popa, R.A. (2016) *Arx: A DBMS with Semantically Secure Encryption*, Technical Report UCB/EECS-2017-111, Electrical Engineering and Computer Sciences University of California, Berkeley [online] https://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-111.pdf (accessed 26 September 2018).

Popa, R.A., Li, F.H. and Zeldovich, N. (2013) 'An ideal-security protocol for order-preserving encoding', *IEEE Symposium on Security and Privacy (SP)*, pp.463–477.

Popa, R.A., Redfield, C., Zeldovich, N. and Balakrishnan, H. (2011) 'CryptDB: protecting confidentiality with encrypted query processing', *SOSP2011: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, Cascais, Portugal, pp.85–100.

Roopaei, M., Rad, P. and Choo, K.K.R. (2017) 'Cloud of things in smart agriculture: intelligent irrigation monitoring by thermal imaging', *IEEE Cloud Computing*, Vol. 4, No. 1, pp.10–15.

Sánchez-Artigas, M. (2018) 'Enhancing tree-based ORAM using batched request reordering', *IEEE Transactions on Information Forensics and Security*, Vol. 13, No. 3, pp.590–604.

Shafagh, H. (2015) 'Toward computing over encrypted data in IoT systems. XRDS: crossroads', *The ACM Magazine for Students*, Vol. 22, No. 2, pp.48–52.

Shafagh, H., Hithnawi, A., Burkhalter, L., Fischli, P. and Duquennoy, S. (2017) 'Secure Sharing of partially homomorphic encrypted IoT data', *SenSys2017: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, Delft, The Netherlands, p. 29.

Shafagh, H., Hithnawi, A., Dröscher, A., Duquennoy, S. and Hu, W. (2015) 'Talos: encrypted query processing for the internet of things', *SenSys2015: Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, Seoul, South Korea, pp.197–210.

Shynu, P.G. and Singh, K.J. (2018) 'Privacy preserving secret key extraction protocol for multi-authority attribute-based encryption techniques in cloud computing', *Int. J. Embedded Systems*, Vol. 10, No. 4, pp.287–300.

Sun, N., Zhu, H. and Wang, W. (2018) 'Fully homomorphic encryption scheme applied to n bit', *Application Research of Computers*, Vol. 35, No. 4, pp.1179–1181.

Thirumalai, C. and Shanmugam, S. (2017) 'Multi key distribution scheme by diophantine form for secure IoT communications', *IPACT2017: processings of 2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pp.1–5.

Tian, X., Huang, B. and Wu, M. (2014) 'A transparent middleware for encrypting data in MongoDB', *IEEE Workshop in Electronics, Computer and Applications*, Ottawa, Ontario, Canada, pp.906–909.

Vermesan, O. and Bacquet, J. (Eds.): (2017) *Cognitive Hyperconnected Digital Transformation: Internet of Things Intelligence Evolution*, River Publishers, Denmark.

Wong, W.K., Wong, K.W., Yue, H.Y. and Cheung, D.W. (2017) 'Non-order-preserving Index for encrypted database management system', *International Conference on Database and Expert Systems Applications*, Springer, Cham, pp.190–198.

Wu, L., Chen, B., Choo, K.K.R. and He, D. (2018) 'Efficient and secure searchable encryption protocol for cloud-based internet of things', *Journal of Parallel and Distributed Computing*, Vol. 111, pp.152–161.

Xie, J., Deng, Y., Min, G. and Zhou, Y. (2017) 'An incrementally scalable and cost-efficient interconnection structure for data centers', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 6, pp.1578–1592.

Xue, K., Li, S., Hong, J., Xue, Y., Yu, N. and Hong, P. (2017) 'Two-cloud secure database for numeric-related SQL range queries with privacy preserving', *IEEE Transactions on Information Forensics and Security*, Vol.12, No. 7, pp.1596–1608.

Yang, Y., Zheng, X., Guo, W., Liu, X. and Chang, V. (2019) 'Privacy-preserving smart IoT-based healthcare big data storage and self-adaptive access control system', *Information Sciences*, April, Vol. 479, pp.567–592.

Zhang, L., Deng, Y., Zhu, W., Zhou, J. and Wang, F. (2015) 'Skewly replicating hot data to construct a power-efficient storage cluster', *Journal of Network and Computer Applications*, April, Vol. 50, pp.168–179.